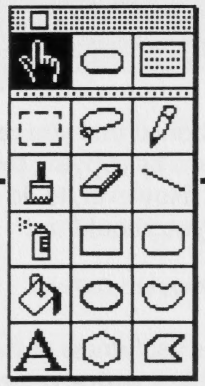
The Apple HyperCard User Group Issue #7 • July 20, 1988

Windoid



A Publication for the Informed HyperCard User

Editor's Corner

By David Leffler

Some of you have been wondering where WINDOID #7 was, well here it is. If it seems to you as if there has been an unusually long time between issues, you're right, it has been. The team has been very busy adding many of the new features you have asked for to HyperCard 1.2.1. However, I think you will find that this issue of WINDOID will have been very much worth waiting for. I have included in here everything I could to give you the information you need to begin using the new version of HyperCard in entirely new and wonderful ways.

Bill and the entire development team have worked hard to give you something really special in this version, and I know you will be as thrilled to use it as we were to assist in its development.

There have been many changes in AHUG since the last issue. I have stepped down as President, and Bryan Carter has emerged as the new President.

Bryan works in the Engineering Support Department here in Apple Cupertino and has done an incredible job of organizing AHUG into a first class User Group.

AHUG was happy to sponsor the first public showing of HyperCard v1.2 on May 5. Bill, Dan, and the HyperCard Team wowed a standing-room-only crowd of 350+ very enthusiastic HyperCard users. The two most admired features were CD ROM/ FileServer compatibility as well as enterInField and returnInField.

On July 8th, author Danny Goodman showed us his soon-to-be released book *Danny Goodman's HyperCard Developer's Guide.* Danny's book takes a three part, indepth look at implementing and designing stacks from a user interface point of view. Danny also hinted about an upcoming update to his acclaimed *Focal Point* time/ project management stack.

Future meetings of AHUG will feature guests from Apple Computer, HyperPro, Personal Training Systems, SuperMac Technologies, HyperAge, MacUser, and MacWorld magazines.

A major AHUG announcement will be made at MacWorld '88 in Boston. Future issues of WINDOID will have more information.

AHUG has a new telephone number for you to call if you would like to obtain information on the AHUG meeting times and forthcoming agenda. The number is 408-974-1707.

Please remember that the purpose of this newsletter is to help you help us to make a quality difference in the world. We provide a unique opportunity for WINDOID readers to contribute directly to the ongoing growth and excellence of Apple's HyperCard™.

There is a form at the back of this, and every, issue of WINDOID that allows you to communicate directly with the HyperCard Team. We really want your input. We want to know what you like, and what you don't like, about HyperCard and what features or suggestions you would like to see implemented in future versions. We give you this unique opportunity because we care about you. For those that wrote to us and participated in HyperCard 1.2.1, we thank you. I believe that you will be very pleased you took the time to make yourself heard.

For those of you that would like additional copies of this WINDOID, or back issues, you must send us a stamped-self-addressed-envelope for each one. We will be unable to send you anything without them. If you send us a very large manilla envelope for all of the issues, please put on enough postage for about fifty (50) pages.

Bryan is putting together a mailing list and will be telling you how to receive regular copies as I publish them. Again, if you like HyperCard and WINDOID, please let us know. You will be glad you did.

If you would like information about AHUG, please send a stamped-self-addressed-envelope to:

AHUG c/o Bryan K. Carter
Apple Computer, Inc.
MS/27-AN
20525 Mariani Ave.
Cupertino, CA 95014

If you have a bug, suggestion, or HyperCard comments use the Form. Or copy the format and Apple-Link™ it to:

HYPERBUG$

In This Issue

- HyperCard Tips
- Four Buttons Properties in Search of a Script
- Show Me
- HyperCard Novice Corner
- Dictionary
- HC 1.2.1 Release Notes
- The Form

HyperCard User Tips

by Phil Wyman

1 In earlier WINDOIDs, I mentioned that you can get a bit - mapped miniature of your Card by copying the card, then doing a CMD-Shift-Paste. However, to do this automatically through HyperTalk, you should try:

```
DoMenu "Copy Card"
type "v" with commandKey, shiftKey
```

2 Here is a short example of the usefulness of the "do" command: If you have a variable x with a value of "field id 1", then you can say in your script,

```
get line 1 of x
```

What you would get in this instance is the words "field id 1", the text that was in the variable x. In order to get what you really wanted, namely the first line of field id 1, you must issue a "do" command like the following:

```
do "get line 1 of " & x
```

3 Be aware that when you create a new card, the openCard message gets sent before the newCard message.

4 Developer tip: Test your stacks at every userlevel. This is important since you will probably develop at userLevel 5, which will work fine until a user is in browse mode. A typical example of this problem is if you have a DoMenu "Copy Card" in your script. This will cause an error at the lowest userLevels which do not have that menu choice. A workaround to this is to, on openstack, put the current userLevel into a variable, and then set userLevel to 5. Make sure you set it back to the previous userLevel when you leave your application.

5 If you try to go to card foo, and there is no card named foo, HyperCard will not make a fuss. However, you can find out that HyperCard has not found the card by checking "the result". "No such card" is "the result " if the card is not found.

```
go card foo
if the result is "no such card"
then answer "Card not found"
```

6 A returnKey handler intercepts the returnKey when it is hit in the message box, whether there's a message in the message box or not. So when you're using a returnKey handler, take into account that the user may well be sending a message through the message box. I found that I had to look to see if the msg was empty before I did my normal returnKey handler, since I didn't want the returnKey handler to execute if the user was sending another message. The same is true for an enterKey handler.

7 When you hit the grave character in HyperCard, you normally "go back" to the last card you were on. If you wanted a grave character, however, you can type Option-Grave-Spacebar in Geneva 12. For fun , try Shift-Option Grave in Geneva 12. (It gives you a little bunny rabbit in the field. With Geneva 14 you get a bird, with Geneva 18 you get a sheep, and with Geneva 20 you get a Macintosh computer.)

8 Here are some button ideas:
This first button idea is a working radio card button. The radio button checks to see if there are any more radio buttons on, since only one radio button should be on at a time. Put the following script in a button. Set the style to radio button and autoHighliting to true in the button info dialog box. The script checks first to see if the button which was clicked on is highlited. If it is highlited, nothing happens, since a series of radio button should always have at least one button on. If the button is not highlited, then this script hilites it and turns off all the rest of the buttons on the card. This script assumes that you only have one series of radio buttons on each card, and that all of them are card buttons.

```
on mouseUp
 if not the highlite of me then
  repeat with x = 1 to the number of buttons
   if the style of button x is "radioButton" then
     if the highlite of button x then
       set the highlite of button x to false
     end if
    end if
  end repeat
  set the highlite of me to true
 end if
end mouseUp
```

The second button idea is a working check box. All the check box does is go on or off. I found it very difficult to figure out that to get a check box to go on or off, you had to click the autohighlite feature in the button info dialog.

9 If your HyperCard home card doesn't look just like it does in the manual, then you need Times 18 font in your system so that the words "HOME CARD" will appear in the proper font.

10 There is an easy way to append to a file in HyperTalk. Read until it is empty, then do your write command.

Four Button Properties in Search of a Script

by Jim Palmer
(with excellent refinements by Clifford Guren)

Setting the text properties of buttons—the **textFont**, **textSize**, **textAlign**, and **textStyle** properties—is currently accomplished via the grace of the message box. You must type a sequence of the **set** command as follows:

```
set textFont of button "Sample Button" to Helvetica
set textSize of btn "Sample Button" to 14
set textStyle of btn "Sample Button" to bold
set textAlign of btn "Sample Button" to center
```

This article describes a script that lets you set the text properties of buttons using the same Text Style dialog box used to set the text properties of fields and paint text. In addition, the script uses some of the new features in HyperCard 1.2.1 , including the **within** operator and many of the new synonyms.

The script has only two parts, (1) a message called **buttonFont** that performs most of the work and (2) a function called **getButton** that returns the name of a button the user chooses with a click.

When you call **buttonFont**, the basic algorithm is as follows:

- **ButtonFont** changes the cursor to the Arrow cursor to indicate that you should choose a button. It then waits for you to click the mouse.
- After you click the mouse, **buttonFont** calls the function **getButton**, which returns the name of the button you have chosen. If **getButton** returns the empty string, you did not click a button, so **buttonFont** exits.
- **ButtonFont** then gets the current text properties of the chosen button and saves them as default settings.
- Next, **buttonFont** locks the screen, chooses the Text tool, and sets the text properties to the default values. It then brings up the Text Style dialog box with each text property set to the value of the default settings (that is, the text properties of the button). You can then use the dialog box to make any changes to the text properties. Note that **textHeight** is not a property of buttons, so its value is not used by **buttonFont**.
- After you finish with the Text Style dialog, **buttonFont** reads the new values of the text properties. It then sets the text properties of the button to these values. Note that if you canceled the Text Style dialog box, the values are just those of the default setting, so the button's text properties are not changed.
- Finally, **buttonFont** chooses the Browse tool and unlocks the screen.

Since no mouse events are sent while the **buttonFont** message is running, the function **getButton** uses HyperCard 1.2.1's new **within** operator to determine whether you clicked a location on the screen that is within the region of a button:

- **getButton** takes one argument, the point where you last clicked (**the clickLoc**).
- It then goes through all the card buttons using **within** to see if the point is inside the rect of any of these buttons. If so, it returns the name of the card button.
- If **getButton** did not return, it then goes through all the background buttons using **within** to determine if the point is inside the rect of any of these buttons. If so, it returns the name of the background button.
- Finally, if **getButton** did not return, it returns the empty string because you did not click a button.

To get the most use of the **buttonFont** message, you will probably want to copy it, and **getButton,** into the script of your Home stack. There are several ways that you might call it. One is to assign it to a function key. For example, putting the following code in your Home stack lets the F7 key invoke the message:

```
on functionKey whatKey
  if whatKey is 7 then
   buttonFont
  else
  --any other function key assignments
  end if
end functionKey
```

If you don't have function keys, you can assign **buttonFont** to the key command Option-T (to relate it to Command-T, the key command that brings up the Text Style dialog box). Because HyperTalk cannot trap for a single

(Con't)

key press, you will have to press Option-T and then the Return key to send the message. This works best if the property Blind Typing from the User Preferences card is set to true so that the message box does not have to be visible when you type Option-T. The following code lets Option-T invoke **buttonFont** (where Option-T is the character **†**):

```
on †
  buttonFont
end †
```

Finally, here are a few things to note about using the **buttonFont** message:

- Buttons in the following script cannot have more than one **textStyle** property; for example, they can be bold or condense, but not bold **and** condense. The Text Style dialog box, however, allows users to choose more than one text style because fields and paint text can have more than one style. If users forget and set more than one style for a button, **buttonFont** only uses the first (or topmost) style.
- The text properties of buttons do not affect the text that is displayed when a button shows an icon.
- The appearance of a given font depends on the fonts currently installed in the System. For example, if a user sets a button's font to Times but does not have Times in his System, an automatic font substitution will occur.
- With HyperCard 1.2.1, stacks can be write-protected. If a stack is write-protected, none of the choices in the Text Style dialog box are active, so you can't actually change the text properties of the button.
- To cancel the **buttonFont** message before you actually select a button, just click where there is no button (or use Command-period).

buttonFont message and its auxiliary function for HyperCard Version 1.2.1

```
on buttonFont
 put empty into btnName
 set cursor to arrow - - pointing cursor
 -- see if the user selects a button:
 wait until the mouseClick
 set cursor to watch
 put getButton (the clickLoc) into btnName
 if btnName is not empty then -- then the user has
 --chosen a button

 -- get all of its current properties:
 put the textfont of btnName into font
 put the textsize of btnName into size
 put the textstyle of btnName into style
 put the textalign of btnName into align

 -- now lock the screen, choose the Text tool,
 -- and set defaults:
  lock screen
  choose text tool
  set textfont to font
  set textsize to size
  set textstyle to style
  set textalign to align

 -- bring up the Text Style dialog box:
 type "t" with commandKey

 -- get the results of the Text Style dialog:
 put the textfont into font
 put the textsize into size
 put the textstyle into style
 put the textalign into align

 -- set the button accordingly:
 set textfont of btnName to font
 set textsize of btnName to size

 -- trap for multiple styles:
  if the number of items of style > 1 then
    put item 1 of style into style
  end if
 set textstyle of btnName to style
 set textalign of btnName to align

  -- clean up:
 choose browse tool
 unlock screen
end if
end buttonFont
```

```
function getButton clickPoint
 -- check all card buttons:
 repeat with i = 1 to the number of buttons
  if clickPoint is within rect of card btn i then
    return the name of btn i
  end if
 end repeat

 -- if it hasn't returned yet, then it wasn't a
card button;
 -- check all background buttons:
 repeat with i = 1 to the number of background
buttons
  if clickPoint is within rect of bg btn i then
    return the name of bg btn i
  end if
 end repeat
 -- if it hasn't returned yet, it wasn't a button:
 return empty
end getbutton
```

ShowMe
A Home Stack Script

by Paul Foraker

I recently had the pleasure of developing a set of 19 stacks for a client. These stacks were moderately complex, including a context-sensitive help system using hidden fields and buttons. It was easy to lose track of which objects we'd already put on the cards, so I wrote this script to add to the handlers in my Home stack script. Now I can simply type "showMe" into the message box in any stack, and HyperCard will come back with a list of all the objects (including hidden objects) on the card I'm looking at and put the list into a card field that the script draws for that purpose.

-- ShowMe
-- 3/24/88 version 0.4

```
-- ShowMe makes a list of all the objects of the current
-- card and background and puts them in a card field named
-- "objects list". Copy this script into your Home stack,
-- then you can type "showme" into the message box while
-- you're looking at any card.
on showMe
 set cursor to 4  -- wristwatch or set cursor to watch in 1.2.1.
 --  "objectNames" will be the variable we'll load up to contain
 -- the names of the objects on the card.
 -- First, the title:
 put "Objects of " & the name of this card & return & return¬
  into objectNames
```

-- BACKGROUND FIELDS

```
-- Then, we get the field names (if any).
if the number of fields <> 0 then
 repeat with i = 1 to the number of fields
 -- in order to tell whether a field is hidden or not, we'll
 -- use a variable called "showing". We'll begin by making it
 -- empty.
   put empty into showing
 -- Now, for each field, we're putting either nothing or "hidden"
 -- into "showing".
if not the visible of field i then put "[hidden]" into showing
 -- The construction "not the visible of field i" resolves to
 -- either true or false depending whether the field is hidden
 -- or visible.
 -- Now we add the field name, whether it is visible or not, and a
 -- return to our variable "objectNames".
   put the name of field i && showing & return after objectNames
 end repeat
end if
 -- Here, we're going to write a "graphic" element to indicate the
 -- end of the list of fields.
put "=====" & return after objectNames
```

(Con't)

HyperCard Novice Corner 2

by Phil Wyman

One of the most powerful features of HyperCard is clicking a button which will go to a card. Even a novice can make such a button. This feature requires no programming, yet can allow you to create useful stacks. This feature can also be called "linking," but that word is way too scary for us novices. "Linking" brings thoughts of relational databases and high powered programmers. However, we can "link" a button to a card with very little effort in HyperCard.

Starting on the Home card click on the left arrow and click on Authoring. Click on the right arrow to go back Home. Click and hold the mouse down on the word "Edit", which is in the menu bar above to your left. We are activating the Edit menu. Drag the mouse down within the menu until the words "New Card" are highlited. Am I going too fast for you? Let up the mouse. We should have created a new card in your Home stack. We should now be on a blank white card. We will now use another menu, "Objects". We learned to use menus in the preceding paragraph. So, click and hold on the word "Objects" in the menu bar at the top of the screen. Drag the mouse down to the words "New button" and release. A new button should now be in the middle of the card.

A couple of miraculous things have happened. One thing is that we are now in the Button tool instead of the Browse tool. We can confirm this by clicking on the "Tools" menu, and you will notice that the button in the middle of the first row is highlited, not the image of the hand (the Browse tool). Let up the mouse. The other great thing that has

(Con't)

-- BACKGROUND BUTTONS

```
-- Using the same routine we did for the background fields:
if the number of background buttons <> 0 then
  repeat with i = 1 to the number of bkgnd buttons
    put empty into showing
    if not the visible of background button i then¬
    put "[hidden]" into showing
    put the name of background button i && ¬
    showing & return after objectNames
  end repeat
end if
-- again, this is the graphic element that indicates the end
-- of the list of background buttons.
put "=====" & return after objectNames
```

-- CARD FIELDS

```
if the number of card fields <> 0 then
  repeat with i = 1 to the number of card fields
    put empty into showing
    if not the visible of card field i then¬
    put "[hidden]" into showing
    put the name of card field i && showing & return¬
    after objectNames
  end repeat
end if
put "=====" & return after objectNames
```

-- CARD BUTTONS

```
if the number of buttons <> 0 then
  repeat with i = 1 to the number of buttons
    put empty into showing
    if not the visible of button i then¬
    put "[hidden]" into showing
    put the name of button i && showing &¬
    return after OBJECTNAMES
  end repeat
end if
```

-- PUTTING THE LIST INTO A CARD FIELD

```
-- Okay, we've got the entire list of objects. Now, we
-- check the list to see if it already contains a card field
-- named "object list". If it does, then we put our list into
-- that field and we're done. If it does not, then we make
-- the new card field on the fly and put our list into it.

if OBJECTNAMES contains "object list" then
 put OBJECTNAMES & return & return into card field "object list"
 show card field "object list"
  exit showMe
end if
```

(Con't)

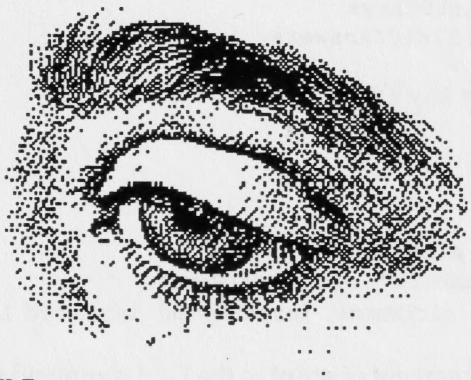
happened is that there are dots moving all around the new button we just created! These are affectionately known as "marching ants." When these "marching ants" are around a button, it tells you that this button is the one we can work on. You can only work on one button at a time, so only one button will have the "marching ants" at any given time.

Now, click and hold on the word "Objects" in the menu bar again. Notice that the words "Button Info" have been darkened and are no longer gray. This means that "Button Info" is active, so let's drag the mouse down until the words "Button Info" are highlited. Let up the mouse.

A dialog box appears on your screen, giving you information about your button. If you start typing, you can give your button a name. There is a button on the dialog box which says "LinkTo...". I want you to click on this button. A small window appears which says: "This Card" ; "This Stack" ; "Cancel". At this point in the "linking" process, we want to go to the card which we want the button to take us to when we click on it. If this is confusing, it should clear up in the next couple of steps.

From the "Go" menu, choose "Home". This will take us to the "Home" card. Click on "This Card" in the small window. You will be returned to the card with your "new button" on it. From the "Tools" menu in the menu bar, drag on to the image of the hand and let up the mouse. This will put you in the Browse tool, and the marching ants around your button should disappear. You have successfully linked this button with your Home card. Now, every time you click this button, you will go to your Home card! Try it.

This linking of a button to a card has many possibilities. For instance, the button we just named could be called "Home", and every time we click on it, we go to our Home card. In the same way, we could make a button called "giraffe", which goes to a card which has a picture and information of a giraffe. Linking with buttons creates a concept of "navigation", which allows users to quickly go where they want to in your stack.



MAKING A NEW CARD FIELD

```
doMenu "new field"  - - makes a new card field
  -- Since new cards are unnamed (unlike buttons, which get the
  -- name "new button"), we'll refer to the field by number. If
  -- we already had, for example, three card fields on the card,
  -- our new field will be number 4. Rather than keep track of
  -- how many card fields we have on this card, we simply refer
  -- to the function, number of, knowing that the number of
  -- card fields has been increased by one. We could write:
  -- get the number of card fields
  -- then use that number to refer to our new field. But,
  -- more directly, we simply put "the number of card fields"
  -- in parentheses where we want HyperCard to count the card
  -- fields, and we end up with:
set name of card field (the number of card fields) to "object list"
  -- Now that we have a name for our new card field, we can refer
  -- to it by its name. Next step is to tell HyperCard
  -- what the properties of the field should be.
set rect of card field "object list" to 0,0,258,342
set style of card field "object list" to scrolling
set textfont of card field "object list" to geneva
set textsize of card field "object list" to 9
  -- Now we put our variable into the field.
put OBJECTNAMES into card field "object list"
  -- The field that we've drawn on the card is pretty big. It
  -- stretches from the upper left corner to about 1/3 of
  -- the way across the card at the bottom. If we want to see
  -- what's underneath the scrolling field, we have to hide
  -- the field. So, we put a handler in the field's script
  -- that lets us do that. This handler hides "the target"
  -- (the object that got the mouseUp message) and also
  -- puts a command into the message box, ready for us to
  -- type a return. That command shows the hidden field again.
put "on mouseUp" & return &¬
"hide the target" & return &¬
"put" && quote & "show" & quote && " && the name of the target" ¬
& return & "end mouseUp" & return into temp
  -- I put the script into the variable "temp" first,
  -- so the next line of HyperTalk code will be simple
  -- and straightforward.
set script of card field "object list" to temp
  -- Now we lock the field so we can click on it.
set locktext of card field "object list" to true
  -- And we're all done, so let's get back to our browsing.
choose browse tool
end showMe
```

DICTIONARY

by Ted Kaehler

Part 1

One of the most useful things a computer can do is look up an item in a dictionary. Like a simple database, a dictionary is a table with two columns.

A script looks up a key in one column and reads off the value from the other column. If the word dictionary does not seem to fit well (it does not contain any definitions), think of a Spanish-English dictionary. There are many different situations in which a quick translation from one thing to another is useful. If you are building an automatic table of contents for a stack, you might want to translate between titles of cards and card ID numbers. In a spread sheet, you might want to translate names of cells into field numbers.

Suppose you have an address stack that contains formal names like "Robert," "William," and "Elizabeth." When you use the Find command, you really want to type nicknames like "Bob," "Bill," and "Liz." Let's write a script that translates the informal names you type into the formal names that appear in an address stack.

A simple way to build a dictionary is to make two lists. The nicknames are separate words in one list, and the formal names are the corresponding words of the other. The lists are stored in variables. The lookup function steps through the words of the nickname list. When it finds the name you asked for, it looks for the word in the same position in the formal name list.

First let's make a function that takes a key and two lists. It looks the key up in the first list and returns a translation from the second:

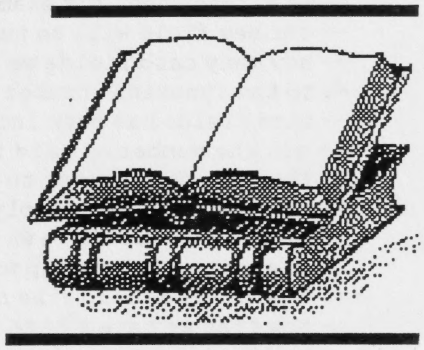
(Con't)

```
function lookup key, listOfKeys, listOfAnswers
  repeat with ii = 1 to the number of words of listOfKeys
     if key = word ii of listOfKeys
    then return word ii of listOfAnswers
  end repeat
  return key -- return the key if it is not in the list
end lookup
```

Here is the way its used:

```
  put "Bob Bill Liz" into nickNames
  put "Robert William Elizabeth" into formalNames
  put lookup(word 1 of it, nickNames, formalNames) into word 1 of it
```

Now, we need to hook this fragment of script to the Find command in our address stack. Let's intercept the Find command and translate the first word of the search key. (Note that the Find command actually has two arguments. The first is a number that keeps track of what kind of Find it is. This number is inserted behind our backs, and all we have to do is know to give it a name.) This script will work in all versions of HyperCard.



```
on find dummy,key
  -- dummy is used privately by HyperTalk
  send "find" && quote & key & quote to HyperCard
  if the result is "not found" then
   put "Bob Bill Liz" into nickNames
   put "Robert William Elizabeth" into formalNames
    put key into newKey
   put lookup(word 1 of newKey, nickNames, formalNames) into
       word 1 of newKey

   if newKey is not key -- if word 1 was translated
   then send "find" && quote & newKey & quote to HyperCard
  end if
end find
```

The script first sends the original Find command directly to HyperCard. If the Find succeeds, do nothing further. If the Find fails, then look up word 1 of the key and substitute it in. If this substitution changes what we are looking for, then search on the new translated version.

Next, we'll refine this example to make it faster and more elegant.

Part 2

Let's build a more powerful dictionary for converting a key into a value. The key may contain more than one word, and there may be multiple keys (synonyms) for the same value. We'll make the dictionary lookup go fast by using the "offset()" function to find the key in the dictionary. Users can decide if the key should be matched completely, or if it need only match the beginning of an entry (like the Find command).

To allow multiple words in keys and values, we can't use space as the separator between entries in the dictionary. The characters { } and | are not used very much and are unlikely to appear in the things you want to put in your dictionary, so we'll use them as separators. A dictionary entry with several keys and a value will be a single piece of text. Every key has a { in before and after it, and the value has a | just before it and } just after it.

(Con't)

Here is the entry that translates nicknames for Elizabeth:

```
{Liz{Beth{Betty{|Elizabeth}
```

Don't include spaces in a dictionary entries unless you want them to be part of a key or a value. Each entry can be on a separate line if you wish.

Let's modify an address stack to translate from nicknames to formal names. We'll keep the dictionary in a global variable called "NickNames". Here is the script that creates the dictonary when you enter the stack:

```
on openBackground
 global NickNames
 put "{Bob{|Robert} {Bill{|William} {Mike{|Michael} " &
 "{Liz{Beth{Betty{|Elizabeth} {Tom{|Thomas}" &
 "{Jim{|James} {Dave{|David} {Joe{|Joseph}" into NickNames
end openBackground
```

Next, let's build a function that does the actual work of looking up a key in the dictionary. The "translate" function takes a key and a dictionary as arguments and returns the value for that key. If the key is not in the dictionary, the function returns the key unchanged.

```
function translate key,dictionary,exact
 put "{" & key into realKey -- {Liz
 if exact is empty          -- look for exact match
 then put offset(realKey,dictionary) into index    -- loc of {Liz. . .
 else put offset(realKey &"{",dictionary) into index -- loc of {Liz{

  -- if the key wasn't found, give back the original key
 if index is 0 then return key    -- Liz
 put char index to (index+200) of dictionary into local
  -- {Liz{Beth{Betty{|Elizabeth} {Tom{|Thomas...
 put offset("|",local) into numStart   -- 17
 put offset("}",local) into numEnd     -- 27
 return char (numStart+1) to (numEnd-1) of local -- Elizabeth
end translate
```

The first half of the translate script sets "index" to the location of the beginning of the key in the dictionary. The second half pulls out a section of the dictionary into the variable "local" in order to work on it. Currently, an entry in the dictionary can't be longer than 200 characters (for keys and value). To allow longer entries, change the "200" to a bigger number. The script finds the | at the beginning of the value part of the entry. Then it finds the next } at the end of the entry. It returns the characters from the beginning to the end of the value it found.

The third argument is optional. If you leave it off, the first argument only has to match the beginning of a key in the dictionary (just like the Find command). If you call translate with anything as the third argument, then the key will only be translated if it matches exactly.

Here are some sample calls:

```
put translate("Betty", NickNames)
put translate("Mik", NickNames,"exact")
```

Mik is not translated to Michael in the second example because we specified an exact match. Remember to put double-quotes around a key if it has any spaces in it. Put the translate function in your home script, and use it with all kinds of different dictionaries.

(Con't)

Here's how an address stack would use translate to convert nicknames to formal names for the Find command:

```
on find dummy,key
  -- dummy is used privately by HyperTalk
 send "find" && quote & key & quote to HyperCard
 if the result is "not found" then
  global NickNames
  put key into newKey
  repeat with jj = 1 to the number of words in key
   put translate(word jj of key,NickNames) into newWord
    put newWord into word jj of newKey
   end repeat

  if newKey is key then beep
   else
   send "find" && quote & newKey & quote to HyperCard
   if the result is "not found" then beep
   end if
 end if
end find
```

This script first tries a normal HyperCard Find on the key. If it is found, do nothing more. If the key is not found, then for each word in the key, translate it through the dictionary NickNames and substitute in the translated word. If the translation (newKey) is the same as the original (key), then none of the words were in the dictionary. Since the first search failed, beep and exit. Otherwise, Find using the new key. If it also can't be found, then beep. (Since we are calling Find from a script, it does not beep by itself.) Notice that the script always replaces a word in the key with its translated value. Since words that are not in the dictionary come back from the translate function unchanged, this works.

Put the translate handler in your Home script to use from many stacks with many different dictionaries. The modified Find handler goes in an address stack. It doesn't belong in the Home script because most Finds have nothing to do with names of people. A quick translation from one set of terms to another is a very useful tool to have in your scripter's bag of tricks. When you're building a stack and suddenly realize that a translation would help, think of these scripts.

American Dictionary

HyperCard version 1.2.1 Final Programmer's Notes

version 1.2.1

By Mike Holm of Apple Computer Co, Inc.
© Apple Computer Co, Inc. 1988

These notes contain information of interest to stack designers and programmers that are not covered in the general HyperCard Version 1.2 Release Notes Stack. Features visible to the average user are described in the Release Notes Stack. The *HyperCard version 1.2.1 Final Programmer's Notes* include the following:

- New HyperTalk synonyms.
- New HyperTalk commands, properties and functions.
- Using "peeking" for fast script review and editing.
- Performance improvements in finding text and navigating stacks.
- User-visible bug fixes.
- Miscellaneous notes on write-protected media and stacks, and related design considerations.

Briefly, HyperCard 1.2.1 includes the following new features:

- On locked stacks, HyperCard 1.2.1 distinguishes between user actions and those of HyperTalk scripts. While users are prevented from making changes to a locked stack (unless the new property userModify is set to true), its scripts can perform most operations. Changes created in locked stacks are temporary—they disappear when the user leaves the card to which changes have been made. Specific restrictions imposed by locked media are explained in detail later in this document.
- Performance and robustness in handling large stacks has been greatly improved. To take advantage of these improvements, stacks created with older versions of HyperCard should be compacted twice with version 1.2.1.
- Pictures on cards and backgrounds can now be hidden and shown. New HyperTalk commands allow stack designers to hide or show art (just like buttons or fields) as part of a HyperTalk script.
- Visual effects work with HyperTalk's new Unlock Screen command. Stack designers may now have a button, field or picture appear or disappear with a visual effect rather than just "popping up."
- Many HyperTalk synonyms, commands and properties have been added or extended.
- HyperCard 1.2.1 allows fast access to HyperTalk scripts by "peeking." Users and designers may quickly view and edit scripts.

New HyperTalk Abbreviations:

- **cd** = card (get the number of this cd)
- **bg** = background (number of cards of this bg)
- **fld** = field (hide fld 3)
- **cds** = cards (number of cds)
- **bgs** = backgrounds (number of bgs)
- **flds** = fields (number of flds)
- **btns** = buttons (number of btns)
- **second** = seconds (wait 1 second)
- **sec** = secs (wait 1 sec)
- **tick** = ticks (wait 1 tick)
- **pict** = picture (hide card pict)
- **grey** = gray (dissolve to grey)

New HyperTalk Messages

- **returnInField**

This message is sent to the field when the Return key is pressed while there is an insertion point or selection in a field. If returnInField is not intercepted by a handler and the insertion point or selection is on the last line of the field, HyperCard will check to see if the field has "autoTab" set to true. If so, HyperCard sends a "tabKey" message to the current field. If the tabKey message is not intercepted by a handler, the cursor will move to the next field. If the insertion point or selection is not on the last line of the field, or if autoTab is false, a carriage return will be inserted in the text of the field.

```
on returnInField -- when the Return key is pressed
 -- prevent insertion of carriage returns anywhere
 --in the field
end returnInField
```

- **enterInField**

The enterInField message is sent to a field when the Enter key is pressed while there is an insertion point or selection in a field. If enterInField is not intercepted by a handler and the contents of the field have been changed, HyperCard sends a closeField message.

```
on enterInField  -- when the enterKey is pressed
 -- find the field contents in another stack
 get me
 go to stack "Address"
 find it
end enterInField
```

New HyperTalk Commands

• Find Whole "string to be found"
Unlike the general Find command, Find Whole uses both word starts and word endings in finding the chosen text. Find Whole also uses space characters in the string. See the HyperCard Version 1.2 Release Notes stack for a description of Find Whole.

• Find String "string to be found"
Find String finds characters specified in a string, ignoring word boundaries.

```
Find String "ple" -- would find the string of
                  -- characters "ple"
```

For strings without spaces, it works the same way as Find Chars. When a string has a space followed by at least three other characters, Find String uses HyperCard's fast search algorithm.

`Find String "ple"` offers the same performance as `Find Chars "ple"`.

`Find String "ple comp"` is much faster because the space and characters in `"ple comp"` invoke HyperCard's fast search.

• hide picture of <card expression>
This command will hide the card picture specified in the card expression. For example:

```
on mouseUp
 hide picture of card 3
end mouseUp
```

• hide picture of <background expression>
This command will hide the background picture specified in the background expression. For example:

```
on mouseUp
 hide picture of background 3
end mouseUp
```

• show picture of <card expression>
This command will show the card picture specified in the card expression. For example:

```
on mouseUp
 show picture of card 3
end mouseUp
```

• show picture of <background expression>
This command will show the background picture specified in the background expression. For example:

```
on mouseUp
 show picture of background 3
end mouseUp
```

• hide card picture
This hides the picture on the current card. For example :

```
on mouseUp
 hide card picture
end mouseUp
```

• show card picture
This shows the picture on the current card. For example :

```
on mouseUp
 show card picture
end mouseUp
```

• hide background picture
This hides the picture of the current background. For example :

```
on mouseUp
 hide background picture
end mouseUp
```

• show background picture
This shows the picture of the current background. For example :

```
on mouseUp
 show background picture
end mouseUp
```

• lock Screen
The lock Screen command performs the same function as "set lockScreen to true." For example:

```
on mouseUp
 lock screen
end mouseUp
```

• unlock Screen [with visual effect]
The unlock Screen command performs basically the same function as "set lockScreen to false". In addition, unlock Screen can take a single visual effect as an argument. The visual effect is invoked as the screen is unlocked ("set lockScreen to false" doesn't invoke visual effects).

```
on mouseUp
 set lockScreen to true
 show card button 3
 unlock screen with dissolve slowly
end mouseUp
```

Note: Visual effects cannot be compounded when using unlock Screen. The visual effects preceding the unlock Screen command shown below will not be executed until a Go command is encountered (or until HyperCard has sufficient idle time to flush them).

```
on mouseUp
 set lockScreen to true
 show card button 3
 visual effect barn door open  -- not invoked by
                               -- unlock Screen
 visual effect dissolve to black -- not invoked by
                               -- unlock Screen
 unlock Screen with checkerboard
end mouseUp
```

In general, visual effects should be as close as possible to the Go command that will use them. Here is an example of multiple visual effects in a single script:

```
on mouseUp
 visual effect zoom open -- will execute with the Go
  -- command
 go to next card
 lock screen
 show card field 3
 unlock Screen with dissolve -- will work with the
  -- unlock Screen command end mouseUp
```

• **select <button expression>**
Chooses the Button tool and selects the specified button. For example:

```
select card button id 78 -- chooses the Button tool
 -- and selects the button
```

This works the same way as the following:

```
choose button tool
click at the loc of bkgnd button 3 -- select
  -- background button number 3
```

This command is especially useful for operations on buttons covered by other objects. It does not work for hidden buttons and works only when the user level is set to authoring (4) or higher.

• **select <field expression>**
Chooses the Field tool and selects the specified field. This command is especially useful for operations on fields covered by other objects. It does not work for hidden fields and works only when the user level is set to authoring (4) or higher.

• **select {before | after} <chunk expression> of <field expression>**
Selects (highlights) text in a specified field as though the user had clicked or dragged over the specified text. "Before" or "after" can be used to place an insertion point at a specific location in a field. A chunk expression can be used to select a range of words or characters in a specified field. For example:

```
select word 3 of field 5 -- highlights the third
 -- word of background field 5
select before char 10 of field 3 — sets the
 -- blinking insertion point between characters 9
 -- and 10 of background field 3
select char 1 to 5 of bkgnd field 1 -- highlights
 -- the first five characters of background field 1
select word 2 to 4 of card field 3 -- highlights the
 -- second, third and fourth words of card field 3
select item 8 of bkgnd field 7 -- highlights the
  -- eighth item in card field 3
  -- remember, items are always separated by commas
```

If a chunk expression of the form "`charStart > charEnd`" is used, a blinking insertion point will be set. For example:

```
select char 10 to 9 of card field 13 -- sets an
 -- insertion point between characters 9 and 10; no
 -- text is highlighted
```

If you "select" past the end of the text in the field, carriage returns will be inserted, and an insertion point will be set after the last return. For example, if a field has only three lines of text in it, and you select line 5, two carriage returns will be inserted and your cursor will be set on line 5 of the field.

```
select line 5 of bkgnd field id 8 -- puts the
 -- insertion point at line 5 inserting any
 -- carriage returns needed to fill empty lines
```

Chunk expressions may also be used to select text in the message box. For example:

```
select word 3 of msg -- highlights the third word of
 -- text in the message box
```

• **select {before | after} text of <field expression>**
Selects all the text in a specified field . "Before" or "after" can be used to place an insertion point at the beginning or end of the text in a field. For example:

```
select after text of card field 7 -- sets the
 -- insertion point after the last character of
 -- card field 7
select text of bkgnd field 3 -- highlights all the
 -- text of background field 3
```

• **select <me | target>**
"Select" may take "me" or "target" as arguments for selecting objects or their contents. For example:

```
on mouseUp
 select me -- chooses the appropriate tool and
 -- selects the object when the mouse clicks on it
end mouseUp

on mouseEnter
 select text of me -- selects all the text of a
 -- field when the mouse enters it
end mouseEnter

on mouseUp
 select the target -- chooses the appropriate tool
 -- and selects the object
end mouseUp
```

• **select empty**
Use "select empty" to deselect highlighted text or remove a blinking insertion point.

• **set cursor to <watch | none | hand | arrow | iBeam | plus | cross | busy>**
Alternate cursors may be used via the "`set cursor to <name or number of cursor>`" command, but are shown only for the duration of the handler. The cursor is always reset to the currently chosen tool (Browse, Button or Field) at idle.

```
on mouseUp
 set cursor to watch — show the watch cursor
 repeat until the mouse is down
        go to next card
 end repeat
end mouseUp
set cursor to none — show the HyperCard transparent
  -- cursor
set cursor to hand — show the HyperCard browse tool
 -- cursor
set cursor to arrow — show the Macintosh arrow
 -- cursor
set cursor to iBeam — show the I-beam cursor
set cursor to plus — show the plus cursor
set cursor to cross — show the cross cursor
```

The "busy" cursor is HyperCard's beach ball. The beach ball will turn 1/8 each time it is set, so it spins when set inside repeat loops.

```
on mouseUp
 repeat for 100 times
  set cursor to busy -- show HyperCard's beach
 -- ball cursor (it will spin)
 add 1 to bkgnd field 1
 end repeat
end mouseUp
```

New HyperTalk Functions & Properties:

• **cantModify**

cantModify is a stack property which prevents users from compacting, deleting or changing the contents of a stack. See the HyperCard Version 1.2 Release Notes Stack for a detailed description. For stacks that are not otherwise locked (on a CD-ROM, locked floppy, locked by AppleShare or the Finder), scripts may get and set CantModify. For example:

```
set cantModify of stack "MyStack" to true
 — prevents users from modifying the stack
```

• **userModify**

userModify is a global property which allows the user to type in fields or paint when the stack is locked. Changes made by the user or script will be discarded upon leaving the card. userModify is set to false whenever the user leaves a stack; it is ignored when a stack is unlocked. This example allows a user to type in a specific field on a locked stack:

```
on openField — user can click in the specified
 -- field or press Tab
 set userModify to true
end openField
```

This next example prevents typing in fields which don't contain the openField handler shown above:

```
on closeField — user can click outside the
 -- specified field or press Tab
 set userModify to false
end closeField
 — closeField is only sent when the field contents
 -- have been changed
```

• **cantDelete**

Scripts may now get and set the cantDelete property for stacks, backgrounds and cards. For example:

```
set cantDelete of this cd to true — prevents
 -- deletion of the card
set cantDelete of this bg to true — prevents
 -- deletion of the background
set cantDelete of this stack to true — prevents
  -- deletion of the stack
```

• **showPict**

When showPict is true, the card or background picture is visible. If false, the picture is hidden. Scripts may get or set showPict. For example:

```
get showPict of background 1
if it is "true" then...
```

• **autoTab**

See the HyperCard Version 1.2 Release Notes for a functional description. Scripts may get or set autoTab. For example:

```
set autoTab of field 3 to true
```

• **number of cards of <background expression>**

This function returns the number of cards of a specified background. For example:

```
get the number of cards of background 3
```

• **the foundText**

Returns the characters enclosed by the box after the Find command has executed. For example:

```
find "Will"
put the foundText — puts "Will" or the whole word
 --it was found in (e.g. "William") into the msg box
```

• **the foundChunk**

Returns a chunk expression for where the string was found. For example, if background field 6 contains the phrase "Now is the time," then:

```
find "Now"
put the foundChunk — puts "char 1 to 3 of bkgnd
  -- field 6" into the message box
```

The foundChunk takes the form: `char <number> to <number> of <card | bkgnd> field <number>`.

• **the foundLine**

Returns an expression for the starting line where the string was found. For example, if line 2 of card field 3 contains the result of the Find:

```
find "Tiger"
put the foundLine — puts "line 2 of card field 3"
 -- into the message box
```

The foundLine takes the form: `line <number> of <card | bkgnd> field <number>` . HyperCard recognizes only lines

terminated by a carriage return. A line that wraps and is displayed as two lines is still one line as far as HyperCard is concerned.

• **the foundField**
Returns an expression for the field where the string was found. For example, if the Find command finds the text in the second line of the third card field:

```
put the foundField -- puts "card field 3" into the
  -- message box
```

The foundText, foundChunk, foundLine and foundField will be empty when a Find command fails because an empty string is returned.

• **the selectedText**
Similar to "the selection," returns the text that is currently highlighted but is not a container into which data can be stuffed.

• **the selectedChunk**
Returns a chunk expression for the range of currently highlighted characters. For example, if a five letter word in card field 9 is selected:

```
put the selectedChunk — puts "char 1 to 5 of card
  -- field 9" into the message box.
```

The expression is of the form: `char <number> to <number> of <card | background> field <number>` .

• **the selectedLine**
Returns an expression for the line where the currently selected text is, using the form:
`line <number> of <card | background> field <number>.`

• **the selectedField**
Returns an expression for the field where the currently selected text is of the form:
`<card | background> field <number>.`

• **left of <button expression> <field expression> <window expression>**
Returns item 1 of the rect (left, top, right, bottom) of a specified object. For example:

```
put left of card button id 456 — puts the value of
the left coordinate of the rectangle of card button
  -- id 456
set left of bkgnd field 5 to 20 — sets the left
 --coordinate of the rectangle of background field
 -- 5 to 20 pixels from the left edge of
 -- HyperCard's window
put left of the card window into MyVar — puts the
 --left coordinate of the HyperCard window into the
 --local variable "MyVar"
get left of tool window — returns the left
 -- coordinate of the tool window
```

"`Set <left | top | right | bottom> of <expression> to <number>`" will move the object so that the specified parameter is at the specified coordinate. It is also much faster than using "`set loc of <expression>.`"

HyperCard's card window uses global coordinates from the Macintosh screen where 0,0 is the top left corner of the screen the menu bar is in. HyperCard windows (card, message, tool and pattern) and objects (button, field) and the Loc (mouseLoc, clickLoc) use local coordinates based on HyperCard's window where 0,0 is the top left corner of HyperCard's card window. HyperCard buttons and fields set with negative coordinates are preserved, but cannot be seen or get mouse clicks.

• **top of <button expression> <field expression> <window expression>**
Returns item 2 of the rect of a specified object.

• **right of <button expression> <field expression> <window expression>**
Returns item 3 of the rect of a specified object.

• **bottom of <button expression> <field expression> <window expression>**
Returns item 4 of the rect of a specified object.

• **topLeft of <button expression> <field expression> <window expression>**
Returns items 1 and 2 of the rect (l, t, r, b) of a specified object. This is the top left corner of the object.

• **bottomRight of <button expression> <field expression> <window expression>**
Returns items 3 and 4 of the rect of a specified object. This is the bottom right corner of the object.

• **botRight of <button expression> <field expression> <window expression>**
botRight is an abbreviation for bottomRight.

• **width of <button expression> <field expression> <window expression>**
Returns the width of the specified object. For example:

```
put width of card button id 456 — puts the width of
  -- card button id 456
set width of bkgnd field 5 to 90 — sets the width
 -- of background field 5 to 90 pixels
put width of the card window into ABC — puts the
 -- width of HyperCard's window into the local
 -- variable "ABC"
```

When the script changes the width of the specified object, HyperCard preserves the loc (center coordinate) of the object, divides the specified width by 2, sets the left to the dividend and sets the right to (left + the specified width). For example:

```
get the width of card button 18 — put the width of
 -- the button into it
set width of card button 18 to it + 20
 — set left to (item 1 of loc - ((it + 20)/2))
```

When the width of a button or field is changed by an odd

amount, HyperCard rounds the dividend down and adds the remainder (1) to the right coordinate.
When the width of a button or field equals 0 or goes negative, the object continues to exist, but cannot be seen or get mouse clicks.

• **height of <button expression> <field expression> <window expression>**
Returns the height of the specified object.
When the script changes the height of the specified object, HyperCard preserves the loc (center coordinate) of the object, divides the specified height by 2, sets the top to the dividend and sets the bottom to (top + the specified height). For example:

```
get the height of bkgnd field 11 -- put the height
 -- of the field into it
set width of bkgnd field 11 to it + 36
 -- set top to (item 2 of loc - ((it+36)/2))
```

When the height of a button or field is changed by an odd amount, HyperCard rounds the dividend down and adds the remainder (1) to the bottom coordinate.
When the height of a button or field equals 0 or goes negative, the object continues to exist, but cannot be seen or get mouse clicks.
Neither the height nor width of HyperCard's window objects (card, message, tools, pattern) can be changed.

• **the screenRect**
Returns the rect of the screen being used (left, top, right, bottom) in global coordinates. When there is more than one monitor, the function returns values for the screen HyperCard's menu bar is in.

• **the long version [of HyperCard]**
Returns the HyperCard version number in the standard Macintosh version resource (see *Inside Macintosh* for explanations of the format). In HyperCard 1.2.1:

```
put the long version -- puts "01208000" into the
 -- message box.
```

• **the version of <stack expression>**
Returns a five item string listing:
1. the version of HyperCard which created the stack.
2. the version of HyperCard last used to compact the stack.
3. the oldest version of HyperCard which changed the stack since it was last compacted.
4. the version of HyperCard which last changed the stack.
5. most recent modification date of the stack (in seconds)

Note: This item is only updated when the stack is closed, not as changes are made. To confirm a change to the modification date, leave the stack, then reopen it and check item 5 of the long version of the stack.
Items 1 to 4 will be set to 00000000 if the version of HyperCard is less than 1.2.1.

For a stack created with HyperCard version 1.1, then edited and compacted with version 1.2.1 things would look something like this:

```
put the version of stack "old stack" -- would put
 --"00000000,01208000,01208000,01208000,2660687462"
  -- into the message box
```

Scripts can convert item 5 into a date and time to determine when the stack was last modified, or check items 1 through 4 to find out whether it had been created, changed or compacted with a version of HyperCard older than 1.2.1.

New HyperTalk Operator:

• **within**
Takes the form: `<point expression> is {not} within <rect expression>` and tests whether a point is within a specific rectangle. For example:

```
on mouseUp
 -- tests the mouseLoc and beeps if it is not
 -- inside the button rect
 wait until the mouseLoc is not within rect of me
 repeat until the mouseLoc is within rect of me
   beep
 end repeat
end mouseUp
```

XCMD Improvements

If an XCMD is called with more than 16 parameters, an error message will appear.

outArgs[1] is guaranteed to be NIL if it is not a valid handle. In previous versions of HyperCard, XCMDs had to check for a result of xresFail to know if you had an invalid handle.

Errors from XCMDs no longer present dialogs to the user. As before, a result of xresFail informs the XCMD that an error has occurred. XCMDs are now free to try evaluating invalid expressions or sending messages that no one receives. They can tell from the result field whether they succeeded or not.

XCMDs can continue running even after an error has occurred. Previously, once an XMCD had evaluated an invalid expression, sent a message that was not caught, or otherwise caused an error, all subsequent callbacks would do nothing. Subsequent callbacks continue to work in HyperCard 1.2.1.

Switching between Browse, Button and Field tools:

Rotating quickly between tools:
- Command-Tab-Tab (in rapid succession) chooses the Button tool.
- Command-Tab-Tab-Tab (in rapid succession) chooses the Field tool.

Command-Tab chooses the Browse tool (as always).

Peeking at Scripts:
The user level must be set to Scripting to "peek" at scripts.

In the Browse tool:

Command-Option mouse click – Users can see scripts for any button by holding down the Command key and Option key, then clicking the mouse on the button they want to "peek" at. Once the script is visible, the script editor may be closed quickly by holding down Command-Option, then clicking the mouse or pressing any key. If the script has been changed, HyperCard will show an alert asking whether the changes should be saved.

Command-Option-c, -b, -s. While holding down the Command and Option keys, pressing "c", "b", or "s" will bring up the card, background or stack scripts respectively. Once the script is visible, the script editor may be closed quickly by holding down Command-Option, then clicking the mouse or pressing any key. If the script has been changed, HyperCard will ask whether the changes should be saved.

Shift-Command-Option-click lets you peek at all field and button scripts (except hidden ones).

In the Button Tool, Command-Option-click lets you peek at button scripts (except hidden buttons).

In the Field Tool, Command-Option-click lets you peek at field scripts (except hidden fields).

Performance and Other Improvements:

• If there is a selection in a locked field, any typing or pasting goes to the message box. The highlighted text in the field will be deselected. A blinking insertion point in the field will be cleared.
• When a handler locks the screen, a number of changes will no longer be visible to the user:
• The cursor won't change when tools change.
• The pattern palette won't change when new patterns are chosen.
• The tool palette won't change when new tools are chosen.
• The menu bar doesn't change when menu choices are made (for example, when switching in and out of painting tools).
• The card window title bar doesn't change when the handler goes to another stack.

When the screen is unlocked, changes will be updated.
• Using the Go command to go to a specific card id is much faster in HyperCard 1.2.1. To take advantage of this, the stack must be compacted with version 1.2.1.
• Using the command `"Go first | next | last | any card of background "foo""` is much faster in HyperCard 1.2.1. Again, the stack must be compacted with version 1.2.1.
• Switching from the Browse, Button or Field tools to the Painting tools is much faster.
• The container "the selection" will now hold 32K of data.
• For operations on fields, "me" can now be used as a container. In the past, "me" always referred to the object — not to its contents. In HyperCard 1.2.1, "me" can refer to the object or its contents.

Version 1.0.1 and 1.1:

```
get the name of me -- returns name of the the object
put "Fred" into me — is incorrect syntax
put me into MyVar — is incorrect syntax
```

Version 1.2.1:

```
get the name of me -- returns name of the object

put "Fred" into me -- or
put "Fred" into line 3 of me
 -- puts the string "Fred" into the field (if "me"
 -- refers to a field or variable)

put me into MyVar
 — puts the contents of the field into the
 --variable MyVar
```

• In the past (for operations on fields) "the target" always referred to the object — not to its contents . In HyperCard 1.2.1, "target" refers to the contents of "the target". "The target" continues to refer explicitly to the object itself.

Version 1.0.1 and 1.1:

```
put the target — puts the name of the object that last
received a message
put target into MyVar — is incorrect syntax
put "Fred" into target — is incorrect syntax
```

Version 1.2.1:

```
put the target — puts the name of the object that
 -- last received a message

put target into MyVar
 — puts the contents of the field into the
 -- variable MyVar.

put "Fred" into target — puts the string "Fred"
 -- into the field.
put value of the target -- is equivalent to "put
 -- target"
```

Note: These changes apply only to operations on fields, not to buttons, cards, backgrounds or stacks.

• Clicking the mouse in a text field while holding the Shift key down does the following:
When there is already text in the field: If the mouse click is below the last line of characters the insertion point is set after the last character in the field . If the mouse click is higher than the last line of characters, the text from the mouse location to the last character will be selected.
If the field is empty, the insertion point will be set at the beginning of the first line of the field.
In versions 1.0.1 and 1.1 of HyperCard, clicking the mouse below the last line of a field inserts Return characters in the empty lines above the location of the mouse click.

- Under MultiFinder 6.0, HyperCard will switch to other applications using HyperTalk's "Open" command, if the desired application is already open.

- When the message box is empty or has spaces in it, HyperCard no longer complains when the Return key is pressed.

- The setGlobal glue routine will now create a global variable using the name you specify if it can't find an existing global with that name.

- If paint images overlapping a field are transparent, the card will print at high quality on the LaserWriter (not 72 dpi).

- The "Open Stack ..., New Stack..." and "Save a Copy..." menu items are now available while a painting tool is selected.

User Visible Enhancements:

- When text has been selected in one field, and the user clicks on another, locked field, the selection is now maintained.
- Tabbing into a field will now select up to 32K (or the entire contents of the field).
- When deleting a card, openCard and closeCard mes sages are no longer sent.
- In the "Stack Info..." dialog, the size of the stack now includes the resource fork.
- When "Copy Current Background" is not checked in the "New Stack..." dialog, HyperCard no longer copies the resource fork of existing stacks with the same name.
- Report printing on the LaserWriter and ImageWriter II no longer occasionally skips pages, prints random data or incorrectly wraps words in mailing labels.

Miscellaneous Notes:

Write Protected (Locked) Media and Stacks
There are five ways to lock or write-protect a stack:

1. Put it on a CD-ROM.

2. Locked floppy disk—setting the write-protect tab on a floppy disk.

3. Putting the stack into a read-only folder under AppleShare, or limiting AppleShare access privileges for the stack itself.

4. Checking the "locked" box in the stack's Get Info window in the Finder.

5. Setting cantModify to true in a script or by checking the Can't Modify box in the Protect Stack dialog (however, setting cantModify to true in a stack is not sufficient to allow multiple user access).

HyperCard detects the first four conditions and automatically sets cantModify and cantDelete to true. HyperCard shows a small padlock icon after the rightmost menu item when a stack is locked.

When a stack is locked and userModify is false, certain menu items are grayed out :

File	Edit	Objects
Compact Stack	Cut	New Button
Delete Stack...	Paste	New Field
Import Paint...	Clear	New Background
	New Card	
	Delete Card	
	Cut Card	
	Text Style	

Scripts can determine whether the first four locking conditions are preventing permanent changes by testing cantModify. For example, a script can attempt to set cantModify to false, then check its state. For example:

```
on mouseUp
 set cantModify of this stack to false
 get cantModify of this stack
 if it is true then ... -- the stack is on locked
 -- media
 else -- the stack isn't on locked media, and can
 -- be modified...
 end if
end mouseUp
```

If the cantModify property of a stack is false before being placed on read-only media, and the stack is later copied to read-write media, cantModify will remain false.

Design Considerations:

Version Requirements - Using HyperCard 1.2.1's new commands will require the stack to be run using version 1.2.1. If none of the new syntax, synonyms, and commands are used, stacks created with version 1.2.1 will work fine with HyperCard versions 1.1 and 1.0.1. Compacting a stack with version 1.2.1 will neither help nor hinder its performance with other versions.

HyperTalk's function "the version" can be used to determine which version of HyperCard you are running under. For example:

```
if the version < "1.2" then set lockScreen to false
else unlock Screen with dissolve
```

Read-only media and stacks – No special commands are required to read a locked stack. But bear two things in mind when creating stacks that will be locked: First, all changes generated by scripts (such as showing a pop-up field) disappear when any move is made to another card. Second, assume the stack will be unlocked somewhere down the line. For example, if you have a field which pops up with a helpful tip for the user, don't rely on the behavior of the locked stack to hide the field when the user is through with it. *Clean up after yourself.*
Here's a list of things which cannot be done by either users or scripts on locked media:

- make new cards
- delete, cut or paste cards
- make new backgrounds
- compact the stack
- delete the stack
- edit patterns
- change the name of a stack
- change any scripts
- sort the stack

Changes to locked stacks are discarded by:

- choosing New Stack..., Open Stack..., Save a Copy..., Print Stack... or Print Report... from the File menu
- toggling to or from the background
- sublaunching another application (using the open command)
- leaving the current card
- opening another stack
- quitting HyperCard

When cantModify is true, setting it to false discards any changes. When cantModify is false, setting it to false saves recent changes to the stack then sets the property.

"Print Stack..." rotates through the cards in the stack while generating the print job, thus any changes to a specific card in a locked or write-protected stack will be discarded before printing.

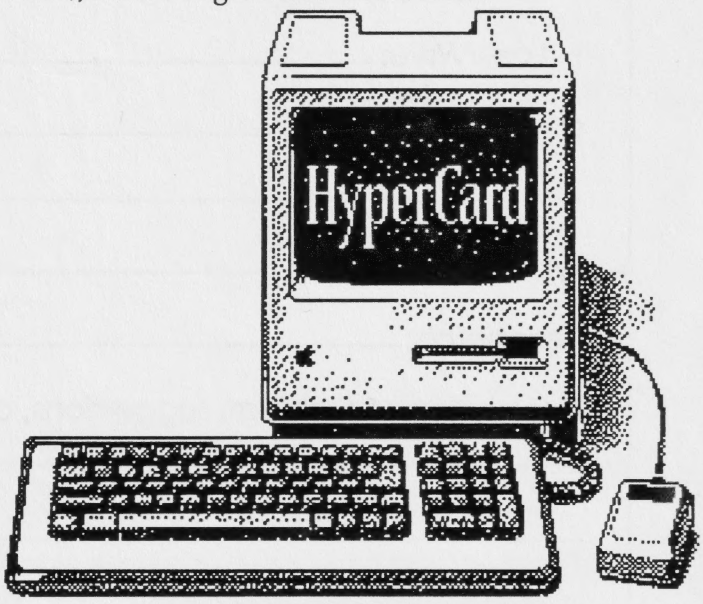
Multiple Users – Several users may read a stack if it's on a CD-ROM, set for read only access, or locked by the Finder. Setting cantModify to true in a stack is not sufficient to allow multiple user access. If one user opens a stack with read-write access (even with cantModify set to true), no other users can open it.

Hidden pictures –(Card or background) remain invisible until the user attempts to edit them. At that time, HyperCard shows an alert asking whether the picture should be shown. Scripts may make changes to a picture without showing the picture (no alerts will appear).

Copying Text and Graphics – Users may set an insertion point in a field to select or copy text, but generally can't type into fields on locked stacks. The selection tools may be used to select or copy graphics from a locked stack.

Using userModify – If a script sets userModify to true, users can type in fields or paint on the card or background. UserModify's default setting is false, and it is reset whenever you close or open a stack. (If you want to save the information the user entered, store it in a global variable.)

Printing – Cards on a locked stack can be printed in the usual way. If a card image is temporarily altered by a script or user, the image displayed on the screen will be printed (when "Print Card" is selected from the File menu), not the original card on the disk.



Windoid Staff

Editor: David Leffler
Copyeditor: Sandy Tompkins
Writers: Members of the HyperCard Team
Layout & Design: Sandy Tompkins

MAKE A DIFFERENCE IN THE WORLD

Tell HyperCard

Date: _____ Phone: _____

Name: _____

MailStop or Address: _____

Versions of:

a. HyperCard _____

b. Associated software _____

_____

c. System _____

d. Finder _____

e. ImageWriter _____

f. LaserWriter _____

g. Any others _____

_____

Type of Macintosh: _____

Peripherals: _____

Description of problem, suggestions, or comments:

_____

_____

_____

_____

_____

_____

please write on the back if necessary

SEND TO:

AHUG
c/o David Leffler
Apple Computer, Inc.
MS/27-AQ
20525 Mariani Ave.
Cupertino, CA 95014
or
copy the format and
Apple-Link™ it to
HYPERBUG$